Interaction of Data Bases and Graphical Interfaces in Civil Engineering

Dipl.-Ing. Irina Biltchouk, Technische Universität Berlin (biltchouk@gmx.de)
Prof.Dr.Dr.h.c.mult. Peter Jan Pahl, Technische Universität Berlin (pahl@ifb.bv.tu-berlin.de)

Summary

Applications for civil engineering tasks usually contain graphical user interfaces for the engineering processes. Persistent objects of the applications are stored to data bases. The influence of the interaction between a graphical user interface and a data base for the development of an civil engineering application is investigated in this paper. A graphic application for the linear elastic analysis of plane frames, which was previously developed with standard tools of the Java platform, is compared to a redesigned implementation using a generalized data base for persistent objects.

The investigation leads to the following results :

- A strict distinction between persistent and transient objects influences the class structure of an application, in particular the class structure of a graphical user interface.
- The structure of an application depends on the logic for updating of references to persistent and transient graphical objects after an application is read from a file.
- The complexity of the reference management can usually be handled better by just in time referencing associated with String - identifiers rather than by automated referencing associated with Name - identifiers.

1 Introduction

The information sets of civil engineering are traditionally ordered with documents. The advantages and disadvantages of a document - independent structure of civil engineering information are being studied in the DFG project “Investigation of Structures in Information Sets of Civil Engineering”. The following generalized system components are developed for this purpose :

- a data base for objects with persistent identifiers
- a component for composition and decomposition of dynamic relations between objects
- a component for composition and decomposition of models
- a component for visualization of applications
- an updater for modifications of a data base
- a navigator for a data base

For each of these components, an application using that component is compared to a standard Java implementation of the same application in order to evaluate the influence of the new concepts on the structure and functionality of the software and its interfaces.

In this paper, a graphic application (linear elastic analysis of plane frames), which was previously developed with standard tools of the Java platform, is compared to a redesigned implementation using a generalized data base for persistent objects. The differences between the class structures of the two applications are presented and evaluated in section 4. In the redesigned application, a solution with String identifiers for encapsulated objects is compared to a solution with Name - identifiers. The influence of the data type of the identifiers on the complexity of the application is evaluated in section 5.

2 Data base

2.1 Concept

The data base which is used in this paper was presented at the IKM 2003 (Biltchouk and Pahl 2003). Its properties can be summarized as follows :

- Classes using the data base do not require a special definition.
- The storage space of the data base consists of a working space and binary files (Fig.1).
- Each object is addressed in the methods by its name, independent of its place of storage.
- The application developer chooses the file in which the object is stored.
- An object is automatically loaded from file to the working space when it is needed in a method.
- References between objects can be set automatically at a point in time chosen by the application developer.
- The data structure of an application is automatically preserved in the data base.

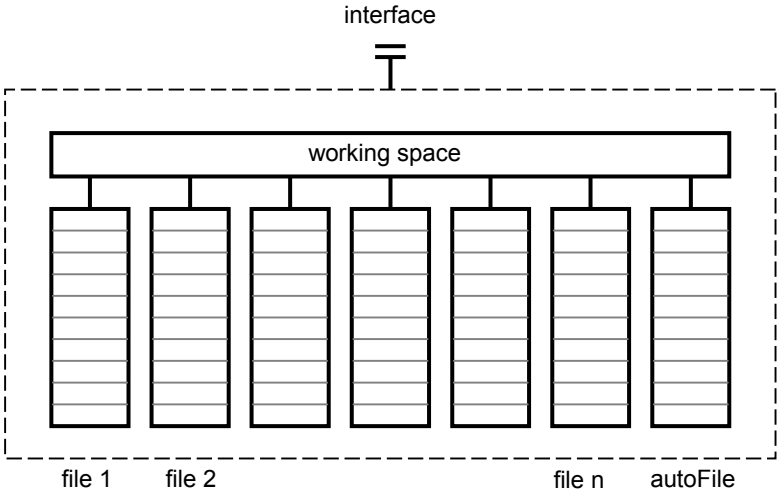

Fig.1 Storage space of an data base

All persistent objects of an application have unique names. An object is called named if its persistent identifier is stored in its body. The named objects interact with the persistent objects of the Java platform, for example collections and graphical objects, which can not be named. Unnamed objects are therefore stored in the data base with handles, i.e. names which are automatically assigned by the base. The reference of an object is obtained by calling getObject(name). Since named objects are stored with special methods of the data base and unnamed objects by Serialization, named objects should be used wherever it is possible.

3 Test case : Linear elastic analysis of plane frames

Figure 2 shows a window for the linear elastic analysis of plane frames. This graphical user interface permits the definition, change and removal of nodes, bars, supports and loads and the specification of their properties with editors.

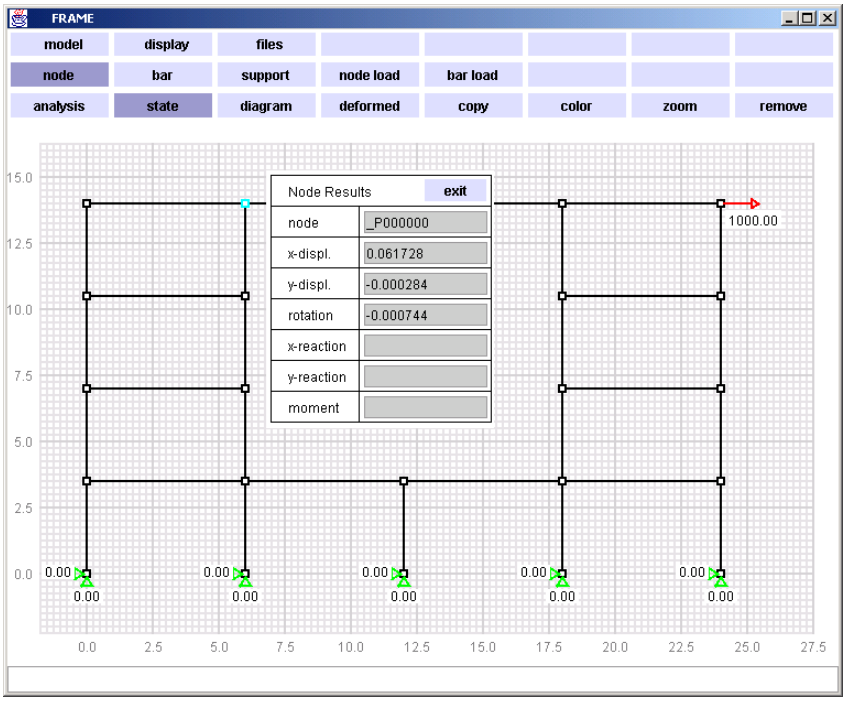

Fig.2 Screen image of the running application

The properties of the structure and its behavior are presented in editors for nodes and bars, in diagrams for the displacement and stress resultants in bars and in diagrams of the deformed structure which can be scaled by the user. The interaction is controlled with menus, control buttons, mouse clicks in the grid of the graphic panel and special key functions.

4 Analysis of the class structure

4.1 Analysis of the standard Java application

4.1.1 Components of the application

The test application for the linear elastic analysis of plane frames has the two main components (Fig.3):

- the product model of the application including the structural analysis
- the visualization model for the graphical frame and the structural frame.

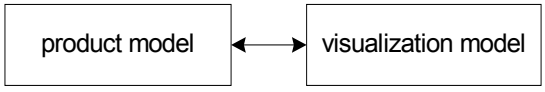

Fig.3 Structure of the standard Java application

The product model describes a plane frame consisting of nodes, bars, supports and loads. It contains the methods for the linear elastic analysis of the frame. The visualization model describes the structure of the menu bar as well the component, function, graphic and message panels and manages the transactions. Typical transactions are the definition, change and removal of nodes, bars, supports, loads. The editors for the properties are automatically offered at suitable locations on the graphical panel. The visualization model also contains methods for the presentation of the structural behavior, in particular the displacements and support reaction at the nodes, diagrams of the deformation and variation of stress resultants in the bars and

overall presentation of the deformed frame. Objects in the two components of the application reference each other.

4.1.2 Storage

The structure of the application is implemented with standard Java tools. The persistent objects are serialized to the file at the end of the session and recovered by deserialization at the beginning of the next session. An object output stream is created to store the objects to the file. The application developer specifies the references of the persistent objects for the stream. An object can be serialized if its class implements the interface Serializable. All non-static and non-transient fields of the object are stored to the file. An object input stream is created to read the objects from the file. The objects are read from the file and recovered in the same sequence in which they were written to the file. The persistent attributes of an object are set to the values which are read from the file. The transient attributes are set to default values.

4.1.3 Reconstruction of the references

The reconstruction of the references is analyzed in the following examples.

The attributes of the graphical panel object contain persistent and transient information. Information like the color of lines or the visibility of loads in the graphical panel is persistent. In contrast, the references of the editors are transient attributes of the graphical panel object : the editors are not serialized. When the attributes of the graphical panel are read from the file and the graphical panel object is reconstructed, the references of the editors must be set to the current values, which in general differ from the values when the graphical panel was stored.

When an application is stored in the file, the check box item which indicates the visibility of the loads in the graphical panel is set to a specific value. The menu bar and the check box item are transient and therefore not stored. The attributes of the graphical panel object including the variable status of the check box item as a persistent variable are written to the file. When the application is read from the file, the current setting of the check box item in the window may differ from the setting when the application was stored. The current menu bar object must therefore know the reference of the reconstructed graphical panel object in order to read the value of the check box item and to set its image accordingly.

Different product and visualization models use the same window as graphical user interface. Therefore the function to manage the setting of the references is a part of the window class. This class contains the list of persistent and transient objects whose references have to be updated after the application has been read from the file.

4.1.4 Conclusions

The analysis of the application class structure leads to the follows conclusions :

- Objects of the graphical user interface contain both persistent and transient attributes.
- The persistent and transient objects reference each other. Therefore the references have to be reconstructed by application logic after an application is read from a file.
- The storage of the persistent objects and the reconstruction of the references are managed by the visualization model.

4.2 Analysis of the application with the data base

4.2.1 Components of the application

The application for the linear elastic analysis of plane frames using the data base (Fig. 4) has two new components in addition to the components of the standard Java application :

- the root for the components of the application
- the data base for the storage of the persistent objects.

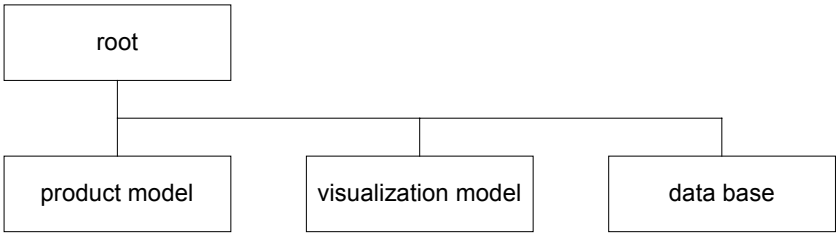

Fig.4 Structure of the application with the data base

The root contains static methods which return the references of the main components of the application, in particular the references of the product model, the visualization model and the data base. The class which implements the methods for the root is called the root class and by convention is named Session. The root class contains methods which are required to adjust the references of the main components of the application after the application is read from the file.

4.2.2 Class structure of the graphical user interface

The application with the data base is structured in such a way that the benefits of the data base management can be utilized for the application. This requires the strict separation of persistent and transient objects. In addition, the distinction between named classes developed in an application and unnamed classes of the platform, whose objects are managed with handles, is of advantage for the application.

Intentional distinction between persistent data dependent on the product model and persistent data independent of the product model contributes to a clean structure of the visualization model (Fig.5).

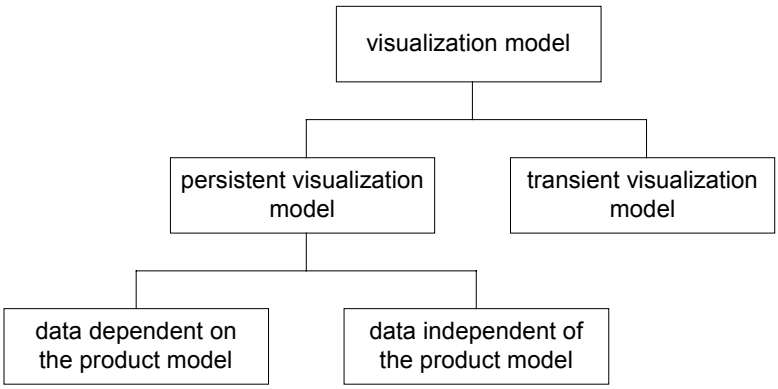

Fig.5 Structure of the visualization model

In the standard Java application, the properties of the graphical panel are attributes of a single object. In contrast, the properties of the same graphical panel in the application with the data base are attributes of following three objects :

- an object of the named class ViewFrame which contains the persistent visualization data dependent on the structural frame.
- an object of the named class ViewPanel which contains the persistent visualization data independent on the structural frame.

- an object of class GraphicsPanel which contains the transient information for the visualization.

4.2.3 Storage

The application is stored in two types of file. The first type consists of the file which is unique to the application and contains the persistent attributes of the data base object such as a last assigned handle, the set of the user identifiers and the list of the open files of the session as well as the names of the product and the visualization model. This file is called the application file. It is read at the beginning and stored at the end of the session. The other files are specified by the user and contain the persistent objects of the product and the visualization models which are stored with methods of the data base. The method storeSession() of the root class calls the data base to write its persistent attributes to the application file. The root class also recovers the data base object with its persistent attributes from the application file before the product and visualization model are read from a file. The names of the product and visualization models which are read from the application file are used to call the data base to read the product and visualization models from a file.

4.2.4 Reconstruction of references

The reference of the Class-object of the root class can be addressed from any method of the application at any point in time. If the product model is read from a file, its reference in working space is recorded in the Class-object of the root class and replaces the reference of the previous product model. If the Class-object of the root class is called to return the reference of the product model it now returns the new reference. It is the responsibility of the application developer to make sure that he always uses the updated references of the product model. The references of the data base and the visualization model are reconstructed analogously.

The reference of a persistent object after storage in a file will differ from its reference before storage. If a method of the application uses a persistent object, the current reference of the object can be obtained from the data base at every point in time by specifying the name of the object.

4.2.5 Conclusions

The analysis of the class structure of the application with the data base leads to the following conclusions :

- The strict distinction between persistent and transient objects influences the structure of the application, in particular the class structure of the graphical user interface.
- The application has a root class whose Class-object can be referenced during a session.
- The root class is responsible for the storage of the persistent attributes of the data base object to the application file and for the reconstruction of the data base object with its persistent attributes which are read from the application file.
- The root class records the references of the main components of the application.
- The methods of the root class implement the logic which updates the references of the main components.
- The methods of the application obtain the current references of the persistent objects of the product and the visualization models from the data base just in time.

5 Identification of encapsulated objects

5.1 Introduction

The data base management differentiates between dependent and independent objects. An object is independent if the application developer calls the data base to write or to read the object. An object B is directly dependent on an object A if an attribute of A is an identifier of B. An object is called encapsulated if it is in one of following forms :

- a dependent named object
- a named or unnamed object which is an element of a set object
- a named or unnamed object which is an element of an array

An object is called open if it is not encapsulated. An open object is an independent named object, a set object or an array.

The open objects are identified in the data base either by a unique name of data type String or by an automatically assigned handle of data type char[]. An encapsulated object is identified either persistently by its unique name or handle, or transiently by its reference in the working space. The data base provides two types of persistent identification for encapsulated objects :

- objects of date type String
- objects of data type Name which contain the String name or handle of encapsulated object and its reference in working space.

If an object is identified by a name or handle, its reference is obtained as return value of the method getObject(name) of the data base. If an object A contains encapsulated objects which are identified with Name - objects, their references are obtained as follows :

- The method setReferences(A) of the data base is called.
- The method uses the reflection concept of Java to find the attributes of data type Name of object A.
- The name or handle in the Name - object is used to read the reference of the encapsulated object from the object map of working space (WSP) as is shown in figure 6.
- The reference is entered in the Name - object.

The method removeReferences(A) sets the references in all Name-objects of A to null.

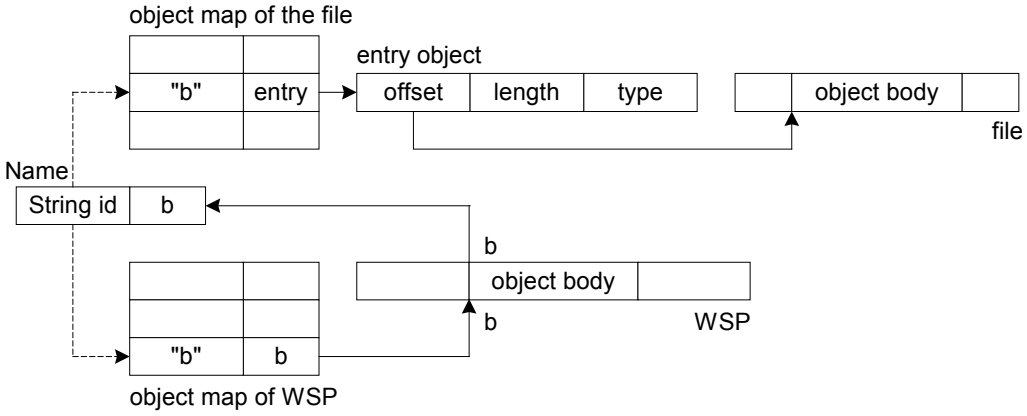

Fig.6 Concept of a Name - object

Assume that an encapsulated object X is moved from working space to a file. Its reference in working space is no longer valid. Assume that the object X is later read from the file. The new reference of X in working space must be entered in all objects which encapsulate X. The choice of the type of the identifier for the encapsulated objects influences the management of their

references at this stage. In the following, three possible concepts for the management of the references of encapsulated objects which are based on the two types of the identifiers are discussed and evaluated.

5.2 Concept A

The encapsulated objects are identified with names of data type String. Before a method of the application accesses an object for the first time, the data base must be called to get its reference. The method getObject(name) of the data base returns the current reference of the object in working space. If the objects is not in working space, the data base management looks for the object in the open files. If the object is found, it is read into working space and its reference is returned to the method of the application. If the object does not exist in the data base, the return value of the method is null.

This concept has the advantage that the correctness of the reference of an object is assured. If the object is moved from working space to a file and then read from the file to working space, the data base gets the correct reference at every point in time. The application developer does not need to control the validity of references if the object is encapsulated at different places in an application. The concept has the disadvantage that the references cannot be set or removed automatically by a method of the data base, because the String-identifier cannot be distinguished from other String-attributes of an object.

5.3 Concept B

The encapsulated objects are identified with Name-objects. If an encapsulated object is created, removed from working space to a file or read from the file to working space, the application developer specifies all open objects which encapsulate the given object. He chooses points in time for setting and removing the references in the corresponding Name-objects by calling the methods setReferences(openObject) or removeReferences(openObject) of the data base. These methods automatically update the references in all Name-objects of the open object. The application developer uses knowledge of the logic of the application to assure that references are correctly updated before they are used.

5.4 Concept C

The encapsulated objects are identified with Name-objects. If an object is created, removed from working space to a file or read from a file to working space, the data base management automatically determines all open objects which encapsulate the given object. The references in the Name-objects of the open objects are automatically updated. This concept has the advantage that the validity of the references in the Name-objects is automatically guaranteed by the data base.

Concept C can be implemented if one is willing to double the required storage space for identifiers (symmetric referencing). Whether such a procedure is advantageous dependents on the properties of the specific application. Since the data base is designed to be independent of the specific application, it has been decided to provide the implementation of concept C in the model management of the overall system. The model management is in parts application-dependent.

5.5 Choice of concept

Each of the concepts A, B and C can be used for the management of encapsulated objects. The three concepts differ in the manner in which references are set and in the degree to which the validity of the references is guaranteed at a given point in time. In concept A the application

developer himself assures the validity of the references by obtaining them from the data base just in time. In concept B the references of groups of objects are set with a single statement. It depends on the specific application whether the sequence of operations is such that the consequences of setting the references at a particular point in time can be foreseen reliably by the application developer. If concept C is used, the data base is responsible for the validity of the references.

The concepts A and B were tested with the redesigned application for the linear elastic analysis of plane frames. The results show that the logic for the determination of the points in time for the automatic transformation of names or handles to references of encapsulated objects is complex and prone to mistakes in big applications. In order to avoid the resulting errors, concept A was chosen for final redesign of the application. Since an object is usually accessed several times after its reference has been obtained, both the programming effort and the execution time required for this method are fairly low relative to the overall requirements for the application. The runtime behavior is not affected significantly by the redesign. The transparency of the class structure and suitability of the software for future expansions are significantly enhanced.

Concept C was not evaluated in the redesign. It will be evaluated in connection with development of the model management.

6 Results of the redesign

The external appearance and functionality of the plane frame analysis application was maintained unchanged in the redesign of the application (Fig.7).

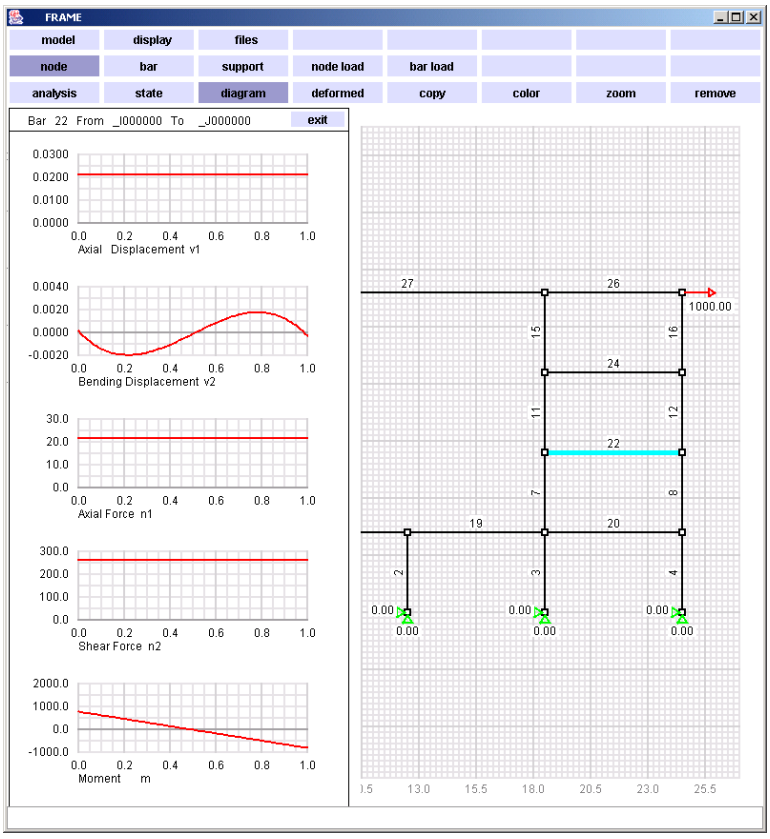

Fig.7 Displacements and stress resultants in a selected bar

The modifications consist in changes of the class structure and in the access methods for the objects of the application. The following examples are typical for the type of changes that were made.

Figure 8 shows a part of the class structure of the redesigned application. Class Session is the root class of the application. Class Frame describes the product model of a structural frame. An object of class Frame contains sets of nodes, bars, supports, node loads and bar loads. The persistent information for the visualization which is dependent on a structural frame is contained in an object of data type ViewFrame. The persistent information for the visualization which is independent of a structural frame is contained in an object of data type ViewPanel. Class Display extends the JFrame of Java and is a graphical frame of the graphical user interface. Class GraphicsPanel contains the transient information for the visualization and manages transactions. The MenuBar is used to select the menus : model, display and files. The toggle buttons for nodes, bars, supports, node loads and bar loads are collected in the ComponentPanel. The functionality of the user graphical interface is controlled with buttons for analysis, report the state of a node or a bar, show diagrams for the displacement and stress resultants in bars and diagrams of the deformed structure, copy a frame component, change a color of lines, zoom and remove in the ActionPanel. The GlassPane protects the content panel of the display while the mouse is dragged to define the zoom box.

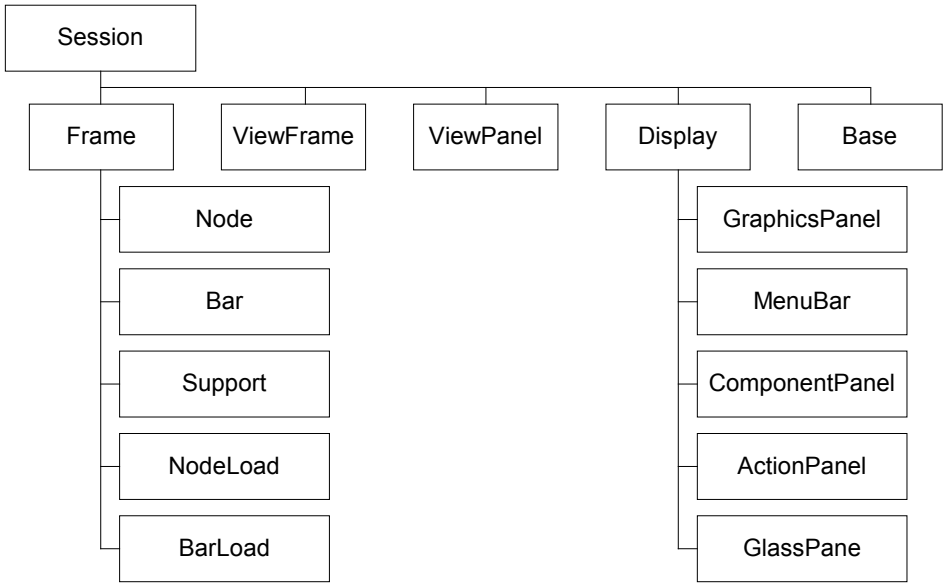

Fig.8 Part of the class structure of the redesigned application

Figure 9 shows a part of the method paintComponent(Graphics g) of the class GraphicsPanel. It is used to illustrate the access to the objects in the redesigned application. The access to the persistent information describing the visibility of supports on the graphical user interface is programmed in line 4. The root class Session is called for the current reference of the ViewFrame which contains the boolean attribute drawSupports. If the value of the attribute is true, the supports are drawn. The support set of the structural frame is accessed in line 5. Lines 7 and 8 illustrate the access to the encapsulated support objects which are identified with String-names in the support set. The root class Session is called for the current reference of the data base. The method getObject(name) of the data base uses the specified name of the support to find its reference in working space. The method draw(g) of the support object is then executed.

```
1     public void paintComponent(Graphics g)
2     {   …
3         // draw the supports
4         if (Session.getViewFrame().drawSupports)
5         {   iter = Session.getFrame().getSupportSet().iterator() ;
6             while(iter.hasNext())
7             {   name = (String)iter.next() ;
8                 support = (Support)Session.getBase().getObject(name) ;
9                 support.draw(g) ;
10        }   }
11        …
12    }
```

Fig.9 Part of method paintComponent in class GraphicsPanel

7 Conclusions and perspectives

7.1 Conclusions

The redesign of the existing application in order to use the functionality of the data base management has let to the identification of three concepts on which future implementations of applications will be based :

1. The class structure of the application must support a clean decomposition into persistent and transient objects.
2. The reconstruction of the persistent objects and the resetting of the references after an application is loaded from a file must be planned systematically. The root concept which is presented in this paper may be compared to the earthing of electrical circuits. It creates a fixed point which permits all methods of an application to address every object at every point in time.
3. Several methods are available for the reconstruction of references after encapsulated objects have been loaded from files. The responsibility for correct references can be shifted to the application developer or to the system components. The choice should be left with the application developer.

7.2 Perspectives

The application for linear elastic analysis of plane frames which is presented in this paper has a single visualization model. The application will be extended to include more than one visualization model for the same product model.

The graphical user interface in the present application is constructed with the usual Java components JPanel, JTextField, JButton etc. The acceleration of the development of the graphical user interfaces by interactive graphic construction, including the establishment of data flow between interface and data base, is under investigation.

8 Acknowledgement

The reported research was conducted in the research project DFG - Gz. PA 162/9-1 “Investigation of Structures in Information Sets of Civil Engineering” of the Deutsche

Forschungsgemeinschaft (German Research Foundation DFG). We wish to thank the foundation for its support.

9 References

Biltchouk, I. and Pahl, P.J. 2003. *Entwicklung einer Datenbasis für Anwendungen im Bauwesen.* In digital proceedings of "Internationales Kolloquium über Anwendungen der Informatik und Mathematik in Architektur und Bauwesen (IKM)". Bauhaus-Universität Weimar. Germany.

Raphael, B. and Smith, I.F.C. 2003. *Fundamentals of Computer-Aided Engineering.* John Wiley & Sons Ltd. West Sussex.